Alim Nizari, Christopher Kassner, John Ye, Thomas Chiu

Project Phase 3: Testing

Before writing unit and integration tests for our game, our first priority was to focus on refactoring our existing code. This consisted of deleting unused files, methods, variables and comments. For example, we deleted an unused Cell.java file, as well as several unused methods in our other files, resulting in a cleaner project with only the necessary files, methods called and variables used.

Next, we began writing unit and integration tests for our game. Most of our game's logic is contained in the Game.java and Board.java files, so first we created two test files – GameTest.java and BoardTest.java. In the BoardTest.java file, we created methods to test if the board's height, width, keys, punishments, enemies, walls and player coordinates are placed correctly on the board. In the GameTest.java, we created a method shouldGameBeOver() to assert that the game's states are correct. Next, we created other various test files for our items, such as BonusTest.java, ExitTest.java, KeyTest.java and PunishmentTest.java. In these files, we created methods to assert that the player is correctly getting and not getting these items. Finally, we created EnemyTest.java and PlayerTest.java files to test if our game's collisions are functioning properly as well as testing the getters and setters for variables including coordinates, velocity, and distances. Essentially, the important interactions between different components of our system such as collisions and game states are all covered by the various tests we've created.

However, tests were not necessary for all of the methods contained in the various files for our game. For example, although the Java Graphics2D class was used for various methods in the Game.java and Board.java files such as drawScore, drawTimer, drawMenu and drawGameOver, it was not necessary to create tests for the methods dealing with graphics and colour. Furthermore, tests for the rendering methods within each object class were not necessary since there are no defined test cases. In addition, upon researching methods for testing graphics, the complexity of the tests includes knowledge that expands beyond the grasp of this course. Due to the fact that not everyone has the same systems and drivers, testing for graphics can be very complicated considering that there may be different possible outcomes when running these tests.

Running our game results in approximately 74% coverage. Our test files such as BoardTest.java, GameTest.java, and the various test files for exits, bonuses, keys and punishments, have 100% coverage. However, it is worth noting that the percentage of coverage in a given test file does not necessarily represent how much of that file corresponding to that given test file is actually being covered. For example, our GameTest.java file has 100% coverage, due to the fact that it only contains the method shouldGameBeOver, which checks the various game states. However, there are many other methods in our Game.java file, such as drawMenu, drawGameOver and drawWin, all using the Java Graphics2D class, as well as methods such as run, start and stop dealing with the game's main loop and other methods checking for key events. These methods related to Graphics2D as well as the methods related to the game's main loop and key events are not tested, therefore the overall coverage of the Game.java file is only 12.0%. Similarly, it was unnecessary to write tests for various methods in the Board.java file dealing with the Java Graphics2D class, hence Board.java only having a coverage of 34.9%. To summarize, 100% coverage is neither necessary nor desirable as many methods included in our game do not need testing. However, the methods related to the core functionality of the game such as movement or collision detection are all covered by various test methods.

During the testing phase, several changes were made to the production code. Namely, a few getter and setter methods, as well as methods for the board to check items such as keys, punishments, bonuses and the exit were added. These were added in to allow for increased communication between files. For example, a setState method was added in the Game.java file, allowing us to set and test the functionality of our game code depending on what state it is currently in. In terms of bugs, there was a bug with the checkExit function where instead of changing the state variable for our game, we were instead changing the GAME_STATE variable to match the WIN_STATE. This resulted in the win screen never being rendered, with just a blank white screen being displayed when trying to exit after collecting all the keys. Otherwise, all of the check methods added to our Game.java file checking for keys, bonuses, punishments, etc, improved the overall quality of our code by increasing modularity. In conclusion, what we've learned from writing and running our tests is that not everything needs to be tested, but crucial aspects of our system should be thoroughly covered and tested. Throughout the third phase, writing tests has been useful to assist in better understanding our code and the different modules our system is broken down into. Making sure that every important aspect of our system is tested and understood was a great learning experience for us and helped enhance our coding skills.